Towards Belief Attribution with Instructor-Facing LLMs

Shlok Shah[0009−0000−8404−3216] and Abdallah Mohamed[0000−0003−0941−2645]

University of British Columbia, Okanagan, Kelowna, Canada
{shlok10@student.ubc.ca,abdallah.mohamed@ubc.ca}

Abstract. Explaining a bug is comparatively straightforward; the harder tutoring problem is understanding why a student wrote it. In CS1, the useful instructional signal is often a student's notional machine, or mental model of program execution, rather than surface correctness alone. We argue that LLMs have an under-explored instructor-facing role: evidence-linked hypothesis generation about plausible student beliefs. Because this role is risk-asymmetric, systems should distinguish structural evidence from semantic inference, normalize abstention, and prioritize specificity rather than broad coverage. To ground the position, we report TRACER, a controlled synthetic probe over 1,200 Java submissions, including 275 examples seeded from 18 misconception variants grouped into ten families. In 5-fold cross-validation, the main condition achieves recall of 0.872 and specificity of 0.848; a label-inclusive ablation raises recall to 0.982 but lowers specificity to 0.774. These results are not evidence of classroom readiness. They show both feasibility and the main safety challenge: models produce plausible belief hypotheses, but over-diagnose clean programs unless evaluation and interfaces reward diagnostic humility.

Keywords: CS1 · notional machines · misconceptions · student modeling · large language models · diagnostic humility

1 Introduction

LLMs are increasingly used in CS1 feedback workflows as on-demand tutors that interpret error messages, explain code, and provide just-in-time help. This paper argues for a complementary use: instructor-facing belief attribution. Instructors often care not only whether a submission is correct, but what patterns of submissions suggest about how students understand execution. The pedagogical question is not merely "what is wrong with this program?" but "what notional machine might have made this program seem reasonable?"

We use *belief attribution* to mean generating plausible hypotheses about a student's latent mental model from observable work. This task differs from bug fixing. A syntax error or failed test can often be described locally; a misconception is a claim about the learner who wrote the artifact. The same code may arise from a stable misconception, a partial understanding, an accidental slip,

or a copied template. Accordingly, an LLM should not be treated as a mind reader or grader. It should be treated as a hypothesis generator whose output is evidence-linked, uncertainty-aware, and verified by instructors.

This short paper makes three contributions. First, it frames instructor-facing belief attribution as a distinct design goal from automated repair. Second, it specifies evaluation requirements for diagnostic humility, especially specificity, clean submission false positives, and abstention. Third, it reports evidence from TRACER, a controlled synthetic probe that shows both feasibility and over-diagnosis risk.

2 Related Work

The concept of a notional machine grounds this work. du Boulay, O'Shea, and Monk describe the learner's need to construct an explanatory model of the "black box" inside programming environments [6]; du Boulay later characterized programming difficulty as involving fragile models of execution [5]. Pea's account of language-independent "bugs" similarly emphasizes systematic conceptual errors, such as learners treating the computer as if it infers human intent rather than executing literal instructions [11]. Sirkiä and Sorva show that many novice mistakes in visual simulation reflect coherent but flawed understandings of assignment, state, and control flow [13].

Automated diagnosis has a long history. PROUST used intention-based diagnosis and a library of buggy plans to interpret novice Pascal programs [8], but such hand-crafted systems were brittle across assignments and programming styles. Later data-driven work compares educator beliefs with student data [4] and clusters code representations to surface misconception-like patterns for expert inspection [12].

LLMs reopen this design space because they can generate fluent explanations and feedback for novice programs [2, 9]. Emerging work also suggests that LLMs can help discover or simulate misconceptions at scale [1, 14]. The safety gap is that LLMs hallucinate [7], and automation bias can lead users to substitute system output for their own vigilance [10]. In educational settings, confident misattributions can also undermine self-efficacy [3]. For instructor-facing belief attribution, the central question is therefore not only whether models can diagnose, but when they should decline to diagnose.

3 Belief Attribution as an Inverse Problem

A *notional machine* is the simplified explanatory machine a course implicitly teaches; a *mental model* is a student's internal approximation of it; and a *misconception* is a systematic divergence from the intended execution model. Inferring a misconception from one program is an inverse problem: the artifact is observable, but the belief is not.

For evaluation, we use an observability axis. A *structural* misconception leaves a visible code signature because the student violates an operational rule of

a language construct, as when a returned value is ignored or 1-based indexing is used. A *semantic* misconception requires inference about an invisible execution-state model, as when a student expects variables to update reactively or treats conditional branches as independent. This binary division is an evaluation convenience, not a claim that observability is inherently binary.

Consider a student who computes a cost formula before reading the inputs it depends on:

```
double y=0, n=0, z=0;
double c = y/n*z;
y = x.nextDouble(); n = x.nextDouble(); z = x.nextDouble();
System.out.println(c);
```

A plausible hypothesis is that the student treats `c` as a live spreadsheet formula rather than a one-time assignment. In TRACER, this is a variant of the *Reactive State Machine* family. Yet the code alone cannot rule out a copied template, ordering slip, or local debugging artifact. A safe tool should present this as one evidence-linked hypothesis, not an asserted fact about the student.

4 Instructor-Facing Hypothesis Generators

An instructor-facing belief-attribution system should produce a small set of plausible hypotheses, each grounded in code spans or behavior. It should separate what is directly supported by the artifact from what is inferred about the learner. It should also aggregate across submissions, because cohort-level patterns can guide instruction without attaching authoritative labels to individual students. By cohort-level analytics we mean prompts for investigation, such as "several submissions compute derived values before reading inputs," not claims that every student in a cluster holds the same internal belief.

This framing changes evaluation. Accuracy and F1 alone are insufficient because they reward coverage even when coverage comes from over-diagnosis. A system designed for diagnostic humility should report false positives on behaviorally correct programs, specificity, and performance by misconception family. Abstention should be a first-class output: a system that declines to diagnose under weak evidence is not failing, it is behaving responsibly.

5 Empirical Grounding: TRACER

TRACER (Taxonomic Research of Aligned Cognitive Error Recognition) is a controlled probe for testing whether LLM outputs about "student thinking" can align with operationalized ground-truth misconceptions. It is not a claim of classroom-ready student modeling, and its quantitative results should be read as illustrative of behavior inside the probe rather than predictive of performance on authentic classroom submissions.

Table 1. TRACER operational taxonomy: 18 prompt-level seeded variants grouped into ten reporting families, split by observability type.

Family	Type	Seeded variants
Reactive State	Semantic	Live derived value.
Anthropomorphic I/O	Semantic	Prompt-as-intent; name-as-intent.
Independent Switch	Semantic	Independent branches; indentation `else`.
Fluid Type	Semantic	String-number coercion; truth coercion.
Teleological Control	Structural	Goal-directed loop update.
Algebraic Syntax	Structural	Chained comparison; equation assignment.
Semantic Bond	Structural	Lasting assignment bond; parameter/alias bond.
Mutable String	Structural	Mutating string method; mutating concatenation.
Human Index	Structural	1-based index; length-as-last-index.
Void	Structural	Ignored-return mutation; print-as-return.

5.1 Human and model roles

The human-controlled parts are the assignments, the notional-machine taxonomy, the misconception seed descriptions, the persona matrix, the prompts, and the black-box I/O tests. The generator LLM instantiates Java submissions under those constraints. Six detector configurations, crossing GPT-5.2, Claude 4.5 Haiku, and Gemini 3 Flash with reasoning and non-reasoning modes, then produce hypotheses from the code and task context. Matching evaluates whether a detector hypothesis aligns with the human-specified seed description. This design creates known ground truth, but it also means the empirical loop is synthetic and model-mediated.

The dataset contains 1,200 synthetic CS1 Java submissions from 300 synthetic students across three assignments, with four submissions per student. The frozen corpus contains 275 misconception-seeded submissions and 925 submissions treated as clean. Generation uses a 4-by-3 persona matrix crossing coding style (minimal, verbose, textbook, messy) with cognitive profile (procedural, mathematical, cautious). These profiles are not claims about real learner types; they are prompt controls to prevent every solution from looking like a single canonical template. Behavioral correctness is checked through black-box I/O tests on compiled Java programs. Each file is evaluated under four prompting strategies by the six detector configurations, yielding 28,800 held-out file–detector evaluations across the five folds.

TRACER operationalizes established notional-machine misconceptions from prior literature as 18 seedable variants grouped into ten reporting families [6, 11, 13]; this grouping is an evaluation instrument rather than a proposed canonical or exhaustive taxonomy. We selected variants that can be injected into short Java programs while preserving a known seed. The quality checks are therefore

Table 2. TRACER summary metrics under 5-fold cross-validation.

Condition	Precision	Recall	Specificity
Label-exclusive matching (main run)	0.577	0.872	0.848
Label-inclusive matching (ablation)	0.511	0.982	0.774

procedural, not psychological: each generated file must compile, exhibit the intended behavior under black-box tests, and contain only the targeted seed. The persona matrix is used only to vary style and problem-decomposition surface form; it is not a theory of real learner types, and TRACER never attempts to recover persona labels from authentic students.

Synthetic data is useful because authentic student data cannot supply an unambiguous ground-truth belief. Real students may hold multiple misconceptions, partially correct beliefs, or errors outside any taxonomy. Injected misconceptions let us measure true positives, false positives, and the observability gap under controlled conditions. This is a reason to use TRACER as a capability probe, not a reason to ignore its synthetic closed-loop limitation.

5.2 Scoring and false positives

TRACER evaluates whether an LLM-generated hypothesis aligns with the injected ground truth. The main condition uses label-exclusive semantic matching, comparing the model's hypothesized belief to the injected student-thinking description while excluding the misconception label text. An ablation includes label text to test whether evaluation shortcuts inflate apparent capability. Current matching scores semantic alignment, not whether cited evidence spans entail the hypothesis.

A false positive is defined relative to the probe design. A clean submission is one that passed behavioral tests and was generated without a seeded misconception. If the detector nevertheless emits a taxonomic hypothesis that matches a misconception family, TRACER counts that as a false positive. This does not prove the code contains no suspicious style; it means the model diagnosed beyond the known seed signal.

Table 2 shows the central trade-off. Label-exclusive matching achieves high recall, suggesting feasibility, but precision remains modest. The label-inclusive ablation nearly saturates recall, but it lowers specificity. In a diagnostic setting, the useful result is not that recall is high; it is that a benchmark can look stronger by coverage metrics while becoming less safe for instructor use.

Figure 1 breaks down performance by family. Structural misconceptions, such as the Void Machine, leave visible syntactic or operational signatures. Semantic misconceptions, such as the Independent Switch or Reactive State Machine, are harder because the artifact may be syntactically valid while reflecting a flawed causal model.

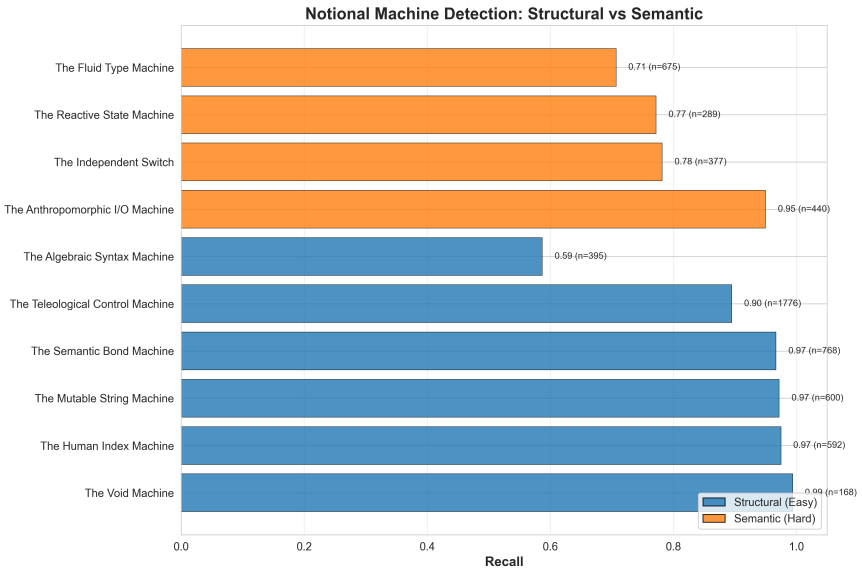

Fig. 1. The observability gap: structural misconceptions leave clearer code signatures than semantic misconceptions.

Table 3. TRACER outcome matrix for the main run. Counts aggregate 1,200 files evaluated with four prompting strategies and six detector configurations.

Ground truth	Correct	Wrong/invented	None	Total
Seeded misconception	5,305 (TP)	520 (FP-wrong)	775 (FN)	6,600
Clean submission	–	3,364 (FP-clean)	18,836 (TN)	22,200

Table 3 decomposes the model's errors. Following the analysis pipeline, aggregate false positives comprise 3,364 invented diagnoses on clean files and 520 wrong-family diagnoses on seeded files; specificity uses only clean-file false positives. Some invented diagnoses may be reactions to style artifacts in the generated code, but under the controlled ground truth they represent failure to abstain. A lightweight abstention analysis used semantic similarity cutoffs of 0.55 and 0.60 plus a 2-of-6 ensemble-agreement rule; in TRACER, ensemble voting reduced false positives by up to 37% while preserving over 99% of true positives. The exact thresholds should not be treated as universal. The important point is that "I do not know" must be an implemented output, not an afterthought.

6 Discussion and Implications

False positives are worse than false negatives in this setting for three reasons. They create epistemic interference, because a student with a correct mental

model may waste effort revising sound reasoning. They create labeling effects, because deficit-oriented explanations can weaken self-efficacy when learners interpret feedback as evidence of inability. They also erode trust: once instructors or students learn that the tool cries wolf, they may discount even valid hypotheses.

For tool builders, diagnosis should be evidence-first and conservative. Interfaces should show code spans and behavior before labels, distinguish direct structural evidence from inferred semantic belief, allow multiple competing hypotheses, and make "no diagnosis" visually normal. For computing education researchers, benchmarks should include clean submissions, report specificity, and audit evidence quality in addition to label alignment. For classroom practice, the safest initial use is cohort-level support for feedback preparation, tutorial planning, misconception analysis, and formative assessment, with instructors inspecting clusters rather than assigning authoritative labels to individual students.

7 Limitations and Future Work

The central limitation is the closed synthetic loop: LLMs help generate the submissions, LLMs diagnose them, and matching compares model output to injected textual descriptions. This can measure whether models recognize controlled misconception patterns, but not whether the same figures transfer to authentic student code. Real students produce messier work, hold multiple misconceptions simultaneously, and make errors outside a predefined taxonomy. The "clean" submissions are behaviorally correct and lack seeded misconceptions, but persona-based generation may introduce unseeded style artifacts that models interpret as evidence. Specificity is therefore conservative with respect to the injected labels, not a final measure of classroom safety. Future work should test transfer to authentic student submissions, evaluate evidence faithfulness, calibrate abstention, and study instructor decision-making in real workflows. For that reason, we do not present the reported recall or specificity values as deployment thresholds. They are stress-test results for a controlled probe. A classroom system would need a second validation stage in which instructors inspect representative cases, compare hypotheses with student explanations, and decide whether the tool changes instructional decisions rather than merely matching a synthetic label.

8 Conclusion

LLMs can help CS1 instruction in more ways than explaining errors or suggesting fixes. TRACER suggests that instructor-facing belief attribution is feasible inside a controlled synthetic probe: models can generate hypotheses aligned with injected notional-machine misconceptions. It also shows why the task requires caution: over-diagnosis is concentrated on clean programs, and semantic misconceptions are less observable than structural ones. The path forward is diagnostic

humility: evidence-first interfaces, normalized abstention, specificity-first evaluation, and human verification. If instructor-facing LLMs are designed this way, they can surface the hidden labor of tutoring while protecting students from confident misattribution.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Al-Hossami, E., Bunescu, R.: McMining: Automated Discovery of Misconceptions in Student Code. arXiv:2510.08827 (2025). https://doi.org/10.48550/arXiv.2510.08827
2. Azaiz, I., Kiesler, N., Strickroth, S.: Feedback-Generation for Programming Exercises with GPT-4. In: ITiCSE (2024). https://doi.org/10.1145/3649217.3653594
3. Bandura, A.: Self-Efficacy: The Exercise of Control. W. H. Freeman, New York (1997)
4. Brown, N.C.C., Altadmri, A.: Investigating Novice Programming Mistakes: Educator Beliefs vs. Student Data. In: ICER, pp. 43–50 (2014). https://doi.org/10.1145/2632320.2632343
5. du Boulay, B.: Some Difficulties of Learning to Program. Journal of Educational Computing Research **2**(1), 57–73 (1986). https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9
6. du Boulay, B., O'Shea, T., Monk, J.: The Black Box Inside the Glass Box. International Journal of Man-Machine Studies **14**(3), 237–249 (1981). https://doi.org/10.1016/S0020-7373(81)80047-9
7. Ji, Z., et al.: Survey of Hallucination in Natural Language Generation. ACM Computing Surveys **55**(12), Article 248 (2023). https://doi.org/10.1145/3571730
8. Johnson, W.L., Soloway, E.: PROUST: Knowledge-Based Program Understanding. IEEE TSE **SE-11**(3), 267–275 (1985). https://doi.org/10.1109/TSE.1985.232210
9. Koutcheme, C., et al.: Open Source Language Models Can Provide Feedback: Evaluating LLMs' Ability to Help Students Using GPT-4-as-a-Judge. In: ITiCSE (2024). https://doi.org/10.1145/3649837.3654178
10. Parasuraman, R., Riley, V.: Humans and Automation: Use, Misuse, Disuse, Abuse. Human Factors **39**(2), 230–253 (1997). https://doi.org/10.1518/001872097778543886
11. Pea, R.D.: Language-Independent Conceptual "Bugs" in Novice Programming. Journal of Educational Computing Research **2**(1), 25–36 (1986). https://doi.org/10.2190/689T-1R2A-X4W4-29J2
12. Shi, Y., et al.: Toward Semi-Automatic Misconception Discovery Using Code Embeddings. In: LAK, pp. 606–612 (2021). https://doi.org/10.1145/3448139.3448205
13. Sirkiä, T., Sorva, J.: Exploring Programming Misconceptions. In: Koli Calling, pp. 19–28 (2012). https://doi.org/10.1145/2401796.2401799
14. Sonkar, S., et al.: LLM-Based Cognitive Models of Students with Misconceptions. arXiv:2410.12294 (2024). https://doi.org/10.48550/arXiv.2410.12294